```
import numpy as np
import random as rand
import cv2

# add padding to image
def clip_filter(image,paddingSize):
    frame = np.zeros((image.shape[0]+int(2*paddingSize)
,image.shape[1]+int(2*paddingSize) ,3),dtype='uint8')
    width = frame.shape[0]
    length = frame.shape[1]

    for i in range(paddingSize,length-paddingSize):
        for j in range(paddingSize,width-paddingSize):
            x= i-paddingSize
            y= j-paddingSize
            frame[j][i] = image.copy()[y][x]

    return frame

def box_filter(image,windowSize=3,imagePaddingSize=0):
    width = image.shape[0]
    length = image.shape[1]

    newImage = np.zeros((width-(2*imagePaddingSize),length-
(2*imagePaddingSize)),dtype='uint8')

    for i in range(0,length-windowSize,1):
        for j in range(0,width-windowSize,1):
            start = (j,i)
            end0 = j+windowSize
            end1 = i+windowSize

            if end0>width:
                end0 = width

            if end1>length:
                end1 = length

            end = (end0,end1)

            buffer = image.copy()[start[0]:end[0], start[1]:end[1]][0]
            buffer_mean = np.mean(buffer)
            newImage[j,i] = buffer_mean

    return newImage



def laplacian_filter(image,mask):

    return weighted_filter(image,mask,weight=1)


def salt_pepper(image, noiseDensity):
```

```
    n_noise = image.shape[0]*image.shape[1]*noiseDensity
    for i in range(int(n_noise)):
        k = rand.randrange(2)
        if k == 0 :
            gray = 0
        else:
            gray = 255

        while(True):
            x = rand.randrange(image.shape[0])
            y = rand.randrange(image.shape[1])
            if len(image.shape) == 3:
                if gray != image[x][y][0]:
                    break
            else:
                if gray != image[x][y]:
                    break

        image[x][y] = gray

    return image

def median_filter(image, windowSize=3,imagePaddingSize=0):

    width = image.shape[0]
    length = image.shape[1]
    image = np.asarray(image , dtype = "int32")
    newImage = np.zeros((width-(2*imagePaddingSize),length-
(2*imagePaddingSize)),dtype='uint8')

    for i in range(0,length-windowSize-1,1):
        for j in range(0,width-windowSize-1,1):
            start = (j,i)
            end0 = j+windowSize
            end1 = i+windowSize

            if end0>width:
                end0 = width

            if end1>length:
                end1 = length

            end = (end0,end1)

            buffer = image[start[0]:end[0], start[1]:end[1]]

            buffer_median = np.median(buffer)

            newImage[j][i] = buffer_median

    return newImage

def weighted_filter(image,mask,weight=1):
    filter = np.zeros_like(image).astype('float32')
```

```
    newImage = np.zeros_like(image).astype('float32')
    mask = mask.astype('float32')
    image = image.astype('float32')

    for i in range(1, image.shape[0]-1):
        for j in range(1, image.shape[1]-1):

            for x in range(mask.shape[0]):
                for y in range(mask.shape[1]):

                    filter[i][j] += mask[x, y]*image[i+(x-1), j+(y-
1)]*weight

            # if filter[i][j] > -50 and filter[i][j] < 50:
            #     # filter[i][j] = 0
            #     newImage[i][j] = image[i][j] - np.abs(filter[i][j])

            #     pass
            # else:
            #     # filter[i][j] += 0
            #     newImage[i][j] = image[i][j] + np.abs(filter[i][j])

            newImage[i][j] = image[i][j] + filter[i][j]



    filter = ((filter-filter.min())/(filter.max()-filter.min()))*255.0
    newImage = np.divide(newImage,(newImage.max()/255.0))
    newImage = newImage.astype('int32')

    return newImage , filter

def mean_square_error(imageSource, imagetarget):
      # the 'Mean Squared Error' between the two images is the
      # sum of the squared difference between the two images;
      # NOTE: the two images must have the same dimension
      err = np.sum((imageSource.astype("float") -
imagetarget.astype("float")) ** 2)
      err /= float(imageSource.shape[0] * imageSource.shape[1])

      # return the MSE, the lower the error, the more "similar"
      # the two images are
      return format(err,'.4f')
```